Formally Verified Argument Reduction with a Fused-Multiply-Add

Sylvie Boldo, Marc Daumas and Ren-Cang Li

Abstract— **Cody & Waite argument reduction technique works perfectly for reasonably large arguments but as the input grows there are no bit left to approximate the constant with enough accuracy. Under mild assumptions, we show that the result computed with a fused-multiply-add provides a fully accurate result for many possible values of the input with a constant almost accurate to the full working precision. We also present an algorithm for a fully accurate second reduction step to reach double full accuracy (all the significand bits of two numbers are significant) even in the worst cases of argument reduction. Our work recalls the common algorithms and presents proofs of correctness. All the proofs are formally verified using the Coq automatic proof checker.**

Index Terms— **Argument reduction, `fma`, formal proof, Coq.**

I. Introduction

Methods that compute elementary functions on a large domain rely on efficient argument reduction techniques. The idea is to reduce an argument x to u that falls into a small interval to allow efficient approximations [1]–[4]. A commonly used argument reduction technique [1], [5]–[7] begins with one positive FPN (floating point number) C_1 to approximate a number $C > 0$ (usually irrational but not necessarily). Examples include $C = \pi/2$ or π or 2π for trigonometric functions $\sin x$ and $\cos x$, and $C = \ln 2$ for exponential function e^x.

Let x be a given argument, a FPN. The argument reduction starts by extracting χ as defined by

$$x/C_1 \quad = \quad \boxed{\qquad \chi \qquad} \boxed{\quad \varsigma \quad}.$$

Then it computes a reduced argument $x - \chi C_1$. The result is exactly a FPN as it is defined by an IEEE-754 standard remainder operation. But division is a costly operation that is avoided as much as possible. Some authors, see for example [1], [3], [7], and

http://www.intel.com/software/products/opensource/libraries/num.htm

introduce another FPN R that approximates $1/C$ and the argument reduction replaces the division by a multiplication

S. Boldo (sylvie.boldo@inria.fr) is with the INRIA Futurs.

M. Daumas (marc.daumas@lirmm.fr) is with the LIRMM, CNRS, UM2 and ELIAUS, UPVD. Supported in part by the PICS 2533 of the CNRS.

R.-C. Li (rcli@uta.edu) is with the Department of Mathematics, the University of Texas at Arlington, P.O. Box 19408, Arlington, TX 76019-0408. Supported in part by the National Science Foundation under Grant No. DMS-0510664 and DMS-0702335 and by the Region Languedoc Roussillon of France.

so that

$$
\begin{aligned}
x \cdot \frac{1}{C} &\approx xR \\
&= z + s \\
&= \boxed{\quad k2^{-N} \quad} \boxed{\qquad s \qquad},
\end{aligned}
\tag{I.1}
$$

where k is an integer used to reference a table of size 2^N. This replacement is computational efficient if u

$$u = x - zC_1 \tag{I.2}$$

is a FPN [8].

Sometimes the computed value of u is not sufficiently accurate, for example if u is near a multiple of C, the loss of accuracy due to the approximation $C_1 \approx C$ may prevail. A better approximation to C is necessary to obtain a fully accurate reduced argument. If this is the case we use C_2, another FPN, roughly containing the next many bits in the significand of C so that the unevaluated $C_1 + C_2 \approx C$ much better than C_1 alone. When equation (I.2) does not introduce any rounding error, the new reduced argument is not u but v computed by

$$v \quad \approx \quad u - zC_2. \tag{I.3}$$

To increase once again the accuracy, the error of (I.5) need to be computed (see Section V), too to obtain v_1 and v_2 exactly satisfying

$$v_1 + v_2 \quad = \quad u - zC_2. \tag{I.4}$$

The last step creates a combined reduced argument stored in the unevaluated sum $v_1 + w$ with $2p$ significant bits

$$w \quad \approx \quad v_2 - zC_3 \tag{I.5}$$

Whether v_1 (or v_1 and w) is accurate enough for computing the elementary function in question is subject to further error analysis on a function-by-function basis [9]. But this is out of the scope of this paper.

The Cody & Waite technique [5] is presented in Figures 1 and 2, where $\circ(a)$ denotes the FPN obtained from rounding a in the round-to-nearest mode. Those are examples when no `fma` is used. The sizes of the rectangles represent the precision (length of the significand) of each FPN and their positions indicate magnitude, except for z and C_1 whose respective layouts are only for showing lengths of significands. The light gray represents the cancellations: the zero bits due to the fact that $|x - \circ(z \times C_1)| \ll |x|$. The dark grey represents the round-off error: the bits that may be wrong due to previous rounding(s).

Figure 1 presents the ideal behavior. Figure 2 presents the behavior when the significand of z is longer. Then, fewer bits are available to store the significand of C_1 in order for zC_1 to be stored exactly. The consequence is a tremendous loss of precision in the final result: as C_1 must be stored in fewer bits, the cancellation in the computation of $x - zC_1$ is smaller and the final result may be inaccurate.

We want to take advantage of the fused-multiply-add (`fma`) instructions. Some machines have hardware support for it, such as machines with HP/Intel® Itanium® Microprocessors [1] and IBM PowerPC Microprocessors, and this instruction will also be added to the revision of the IEEE-754 standard. The current draft can be found at

$$\texttt{http://www.validlab.com/754R/.}$$

It is obvious that some bits of x and zC_1 will cancel each other as z is computed such that $x \approx zC_1$, but it is not clear how many of them will and under what condition(s). Consequently if accuracy calls for $x - zC_1$ to be calculated exactly (or to more than p bits in the significand), how do we get these bits efficiently? This question is especially critical if the working precision is the highest available on the underlying computing platform.

In this paper, we will devise easily met conditions so that $x - zC_1$ can be represented exactly by a FPN, and thus it can be computed by one instruction of the `fma` type without error. This technique is presented in Figure 3. The understanding is the same as in Figures 1 and 2. The cancellation is greater as C_1 can be more precise. The idea is that the rounding in zC_1 is avoided thanks to the `fma`: zC_1, a $2p$-bit FPN, is virtually computed with full precision and then subtracted from x. This subtraction is proved to be exact as $x \approx zC_1$. The fact of $x - zC_1$ being a FPN is used by the library of [1], [7] with no formal justification until [8].

The motivations of this work are similar to those presented in [8] and Section II recalls briefly some useful prior-art from the authors [8], [10]. However, the rest of the paper presents entirely new results. The theorems and their proofs are different from the ones presented in [8]. The changes are necessary to facilitate verification with an automatic proof checker. Moreover, the results have been improved, and are simpler to grasp and new results have been added thanks to this simplification and to a better understanding of the FPNs relationships due to the formal proof.

In a floating-point pen-and-paper proof, it is difficult to be absolutely sure that no special case is forgotten, no inequality is erroneous, and no implicit hypothesis is assumed, etc. All the proofs presented in this paper are verified using our specification of generic floating point arithmetic [10] and Coq proof assistant [11]. This approach has already been proven successful in hardware or software applications [12]–[15]. The drawback is a long and tiresome argumentation versus the proof checker that will ascertain each step of the demonstration. The corresponding scripts of proofs are available online at

$$\texttt{http://www.netlib.org/fp/fp2.tgz.}$$

We indicate for each theorem its Coq name. The developments presented here are located in the `FArgReduct[2,3,4].v` files.

The rest of this paper is organized as follows. Section II recalls theorems on the number of cancelled bits of two close FPNs (extensions of Sterbenz's theorem [16]). In Section III, we present the Coq verified theorem about the correctness of the algorithm that produces z in (I.1) and that satisfies the conditions of the following theorems. The demonstration of the main result, *i.e.* the correctness of the first reduction step, is then described in Section IV. In Section V, we give new algorithms and results about a very accurate second step for the argument reduction. Section VI concludes the work of this paper.

Notation. Throughout, $\ominus$ denotes the floating point subtraction. $\{\mathcal{X}\}_{\mathsf{fma}}$ denotes the result by an instruction of the fused-multiply-add type, i.e., the exact $\pm a \pm b \times c$ after only one rounding. FPNs use p digits, hidden bit (if any) counted, in the significand or otherwise explicitly stated. We denote $\circ(a)$ the FPN obtained from rounding a in the round-to-nearest mode with p digits and $\circ_m(a)$ if we round to m digits instead of p. We denote by $\mathrm{ulp}(\cdot)$ the unit in the last place of a p-digit FPN and $\mathrm{ulp}^{\circ 2}(\cdot) = \mathrm{ulp}(\mathrm{ulp}(\cdot))$. The smallest (subnormal) positive FPN is denoted by λ.

II. EXACT SUBTRACTION THEOREMS

These theorems will be used in Section IV to guarantee that there will be enough cancellation in $x - zC_1$ so that it can be computed exactly by one `fma` type instruction, or equivalently, to assure $x - zC_1$ fits into one FPN.

A well-known property [16], [17] of the floating point subtraction is the following.

> *Theorem 1 (*`Sterbenz`* in* `Fprop.v`*):* Let x and y be FPNs. If $y/2 \le x \le 2\,y$, then $x - y$ is a FPN. This is valid with any integer radix $\beta \ge 2$ and any precision $p \ge 2$.

We extend Sterbenz's theorem to fit the use of a fused-multiply-add that may create a higher precision virtual number whose leading digits are canceled to the working precision as explained in Figure 4: when x and y are sufficiently near one another, cancellation makes the result exactly fit a smaller precision.

> *Theorem 2 (*`SterbenzApprox2`*):* Let x and y be p_1-digit FPNs. If
> $$\frac{y}{1 + \beta^{p_2 - p_1}} \le x \le \left(1 + \beta^{p_2 - p_1}\right)\,y,$$
> then $x - y$ is a p_2-digit FPN. This is valid with any different significand sizes $p_1, p_2 \ge 2$, and any integer radix $\beta \ge 2$.

The proofs are omitted as they appeared in other publications [8], [18]. It is worth mentioning that Theorem 2 do not require $p_1 \ge p_2$ or $p_2 \ge p_1$.

From now on, all FPNs are binary. The underlying machine hardware conforms to IEEE-754 floating point standards [19], [20]. This implies that rounding does not introduce a rounding
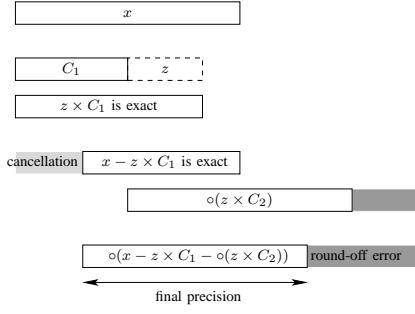
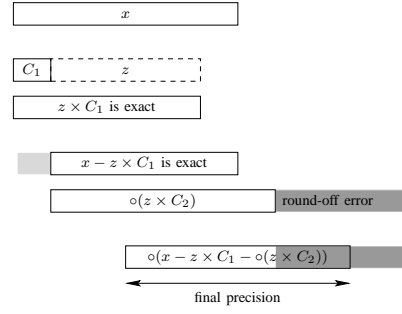Fig. 1. Reduction technique works for z sufficiently small.

Fig. 2. Cody Waite technique fails as z grows, i.e. u is not accurate enough.
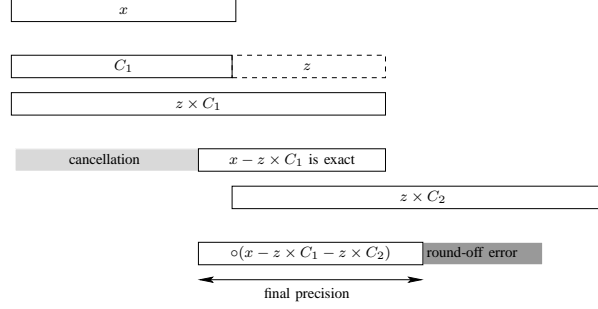

Fig. 3. Argument reduction with exact cancellation in a fused-multiply-add.

error when the exact result is a FPN. Unless explicitly stated, the default rounding mode is *round-to-nearest* with ties broken to the even significand.

III. ABOUT THE ALGORITHM FOR z

The computation of z can be done efficiently as

$$z \;=\; \{xR + \sigma\}_{\mathsf{fma}} - \sigma, \qquad (\text{III.1})$$

where σ is a pre-chosen constant. The technique is adapted from [1, Chap. 10] who used an idea attributed by the author to C. Roothaan in his work for HP's vector math library for Itanium. The explanation is in Figure 5: here we choose $\sigma = 3 \cdot 2^{p-N-2}$ for a z having its last bit at exponent $-N$.

In realizing (I.1), the wanted results are that $z2^N$ is an integer, and that $|xR - z| \leq 2^{-N-1}$. We may also need that the precision needed for z is smaller or equal to $p - 2$. Here is the theorem, verified by Coq.
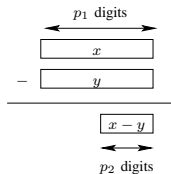

Fig. 4. Extension of Sterbenz's theorem.

> *Theorem 3* (`arg_reduct_exists_k_zH`)*:* Assume
> - $p > 3$,
> - x is a p-bit FPN,
> - R is a positive normal p-bit FPN,
> - $z = \left\{ 3 \cdot 2^{p-N-2} + xR \right\}_{\mathsf{fma}} \ominus 3 \cdot 2^{p-N-2}$,
> - $|z| \geq 2^{1-N}$,
> - $|xR| \leq 2^{p-N-2} - 2^{-N}$,
> - 2^{-N} is a FPN.
>
> Then there exists an integer ℓ satisfying $2 \leq \ell \leq p - 2$ such that
> - $|z2^N|$ is an ℓ-bit integer greater than $2^{\ell-1}$, and
> - $|xR - z| \leq 2^{-N-1}$.

In short, if z is computed as explained and x is not too big, then z is a correct answer, meaning it fulfills all the requirements that will be needed in Theorem 4 in the next section.
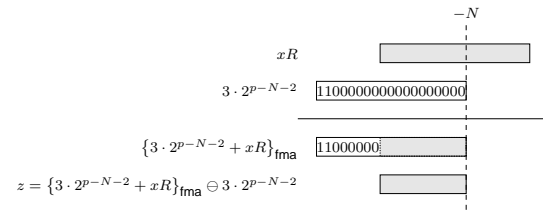

Fig. 5. Algorithm for computing z.

For Intel's double extended precision, this technique is perfectly adapted for range reduction with argument between -2^{63} and 2^{63} when R is in the order of $O(1)$. This argument range coincides with what is in Intel's manual [21] for FSIN,

FCOS, FPTAN and FSINCOS. A quick justification is for $C = 2\pi$ and modest N, say $N = 0$ for an example, $|xR| \lesssim 2^{63}/(2\pi)$ gives $|xR| < 2^{62} - 1$.

For the exponential function, any argument larger than 11356 overflows in the double extended and quad precisions, and $\ell \leq p - 2$ is easily satisfied.

IV. MAIN RESULTS

We now present the conditions under which $x - zC_1$ can be represented exactly by a FPN, and thus it can be computed by $\{x - zC_1\}_{\mathsf{fma}}$ without error. As in Section I, $R \approx 1/C$ and $C_1 \approx C$. We suppose that $C > 0$ and $C \neq 2^j$ for any j.

The idea is the use of a fused-multiply-add that may create a higher precision virtual number that cancels to the working precision. Figure 6 explains the idea: if z is an ℓ-bit integer and the significand of C_1 uses $p-q$ bits, it takes up to $p-q+\ell$ bits to store the significand of zC_1. And as zC_1 and x are near enough, the final result fits into p bits. The notation m_X stands for the significand of X and e_X its exponent.
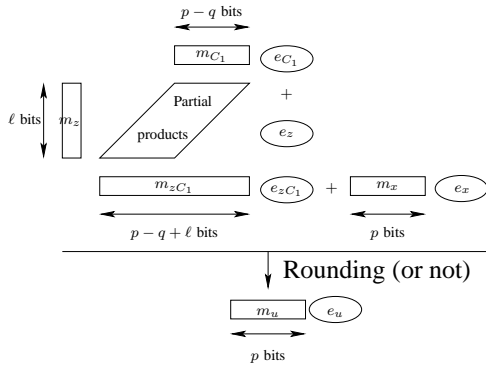

Fig. 6. Fused-multiply-add used to create and cancel a higher precision virtual number.

We want to give enough hypotheses on the inputs to guarantee that $x - zC_1$ will be computed without error.

We define the exponent e_R of R as the only integer such that $2^{e_R} < R < 2^{e_R+1}$. We want to set the q least significant bits of C_1 to zero. Since C_1 should be as accurate as possible, we set $C_1 \approx 1/R$ to the nearest FPN with $p - q$ significant bits. From this, we deduce that $2^{-e_R-1} \leq C_1 \leq 2^{-e_R}$ and that the distance between C_1 and $1/R$ is less than half an ulp (in $p - q$ precision) therefore

$$\left| \frac{1}{R} - C_1 \right| \leq 2^{-e_R-1-(p-q)}.$$

We now define $\delta = RC_1 - 1$, and we deduce a bound on its magnitude from the previous inequalities

$$|\delta| \leq 2^{q-p}.$$

Let z be as defined by (I.1) with the conditions on z and s given there. We assume for the moment that $z \neq 0$. Theorem 2 can be used if we bound $x/(zC_1)$ and its reciprocal by $1 +$

$2^{q-\ell}$. We have the following equalities:

$$
\begin{aligned}
\frac{x}{zC_1} &= \frac{xR}{zRC_1} \\
&= \frac{z+s}{zRC_1} \\
&= \left(1 + \frac{s}{z}\right)\frac{1}{1+\delta}.
\end{aligned}
$$

We recall that $z = k2^{-N}$ and that k is an integer using ℓ bits, and we deduce on the other hand

$$2^{-N+\ell-1} \leq |z| < 2^{-N+\ell}$$

to bound

$$\left| \frac{s}{z} \right| \leq 2^{-\ell}.$$

Rewriting the condition of Theorem 2 and taking advantage of preceding results, we arrive at the point to prove both

$$\frac{1+2^{-\ell}}{1+|\delta|} \leq 1 + 2^{q-\ell}, \qquad (\text{IV.1})$$

$$\frac{1+|\delta|}{1-2^{-\ell}} \leq 1 + 2^{q-\ell}. \qquad (\text{IV.2})$$

Conditions (IV.1) and (IV.2) are checked using functional analysis on polynomials and homographic functions for any permitted value of $A = 2^{-\ell}$. Since z is both a machine number and a non-zero ℓ-bit FPN ($1 \leq \ell \leq p$). From Section III, the algorithm used to produce z implies $\ell \leq p - 2$. We will use a more generic condition:

$$2^{1-p} \leq A = 2^{-\ell} \leq \frac{1}{2}.$$

We will now explain what are the successive requirements to guarantee that both (IV.1) and (IV.2) are fulfilled.

a) Condition (IV.1): We want to guarantee that $\frac{1+2^{-\ell}}{1+2^{q-\ell}} \leq 1 + |\delta|$. The homographic function

$$\frac{1+2^{-\ell}}{1+2^{q-\ell}} = \frac{1+A}{1+A2^q}$$

we want to bound is maximized at $A = 2^{1-p}$ and it is sufficient to check if $(1 + 2^{1-p})/(1 + 2^{1-p}2^q) \leq 1 + |\delta|$. We use the bound on $|\delta|$ and we introduce $B = 2^q$. We have left to prove that

$$(1 + 2^{1-p})/(1 + 2^{1-p}B) \leq 1 - B2^{-p}.$$

This is equivalent to check if the second order polynomial $2^{1-p}B^2 - B + 2 \leq 0$. The inequality is satisfied for B between the two roots $2^{p-2}\left(1 \pm \sqrt{1 - 2^{4-p}}\right)$. Thus it is sufficient to have $B \geq 4$ for all precisions.

b) Condition (IV.2): We want to guarantee that $1 + |\delta| \leq (1 + 2^{q-\ell})(1 - 2^{-\ell})$. We introduce A and B as before, so we have left to prove

$$1 + |\delta| \leq (1 + AB)(1 - A).$$

We assume that $B \geq 4$ from the preceding paragraph. The polynomial

$$(1 + AB)(1 - A) = (1 + 2^{q-\ell})(1 - 2^{-\ell})$$

is minimized at $A = 2^{1-p}$ and it is sufficient to check if $(1 + |\delta|) \leq (1 + 2^{1-p}B)(1 - 2^{1-p})$. From the bound on $|\delta|$, we now have to check if

$$(1 + B2^{-p}) \leq (1 + 2^{1-p}B)(1 - 2^{1-p})$$

which is true for any precision.

This proof is rather long and complex. We therefore verified it in Coq to be sure there is no mistake. It also gives us more precise and sharp hypothesis than if we would do that only by pen-and-paper. All hypotheses have to be clearly written so that the proof can be checked. There is no easy way to say "we assume there is no Underflow" or "that the precision is big enough". This leads to long theorems (at least longer than what we are used to), but precise and correct ones:

Theorem 4 (`Fmac_arg_reduct_correct1`*):*
Assume

- $p > 3$,
- x is a p-bit FPN,
- R is a positive normal p-bit FPN,
- $2 \leq q < p - 1$,
- C_1 is the $(p - q)$-bit FPN obtained by rounding $1/R$ to $p - q$ bits using round-to-nearest mode,
- C_1 is not exactly a power of 2,
- $C_1 \geq 2^{p-q+\max(1,N-1)}\lambda$,
- $2 \leq \ell \leq p - 1$,
- $|z2^N|$ is an ℓ-bit integer greater than $2^{\ell-1}$,
- $|xR - z| \leq 2^{-N-1}$,
- $q \leq \ell$.

Then $x - zC_1$ is a p-bit FPN.

In short, if C_1 is rounded to the nearest from $1/R$ with $p-q$ bits and $q \geq 2$ and z is not too small, then the fma does not make any round-off error.

Automatic proof checking also prompted us that the exact behavior may be difficult to obtain for $z = 2^{-N}$ and x close to $2^{-N-1}R$. This case was excluded in Theorem 4 under the hypothesis that $2 \leq \ell$, but it will be included in the next theorem which focuses on $q = 2$ as this situation leads to C_1 as close as possible from C and thus has more practical value. For completeness and theoretical interest a theorem similar to Theorem 4 but valid for all $2 \leq 2 \leq p - 1$ is presented in the appendix.

Assume $q = 2$ in the rest of this section. When $z = 2^{-N}$, then $x \leq 2C_1 \times 2^{-N}$ as xR is approximated by $z = 2^{-N}$. We can also deduce that

$$\frac{C_1 \times 2^{-N}}{1 + 2^{2-p}} \leq x.$$

When $C_1 \times 2^{-N}/2 \leq x$, Sterbenz's theorem (Theorem 1) can be applied and $x - C_1 \times 2^{-N}$ is representable. If not, then

$$\frac{C_1 \times 2^{-N}}{1 + 2^{2-p}} \leq x < \frac{C_1 \times 2^{-N}}{2}.$$

Since C_1 is a $(p-2)$-bit FPN and not exactly a power of 2 as a p-bit FPN, then C_1 is at least 4 ulps away from a power of 2. This is because as a p-bit FPN, C_1 is worth $2^e \times 1.bb \cdots b00$, where at least one of the b's must be 1; therefore the C_1 that comes closest to a power of 2 is either $2^e \times 1.0 \cdots 0100$ or

$2^e \times 1.11 \cdots 100$. Both are 4 ulps away from a power of 2. This distance and the preceding inequality are enough to guarantee that the exponent of x is the exponent of C_1 minus $N + 1$. After a few computations, we finish with $x - C_1 \times 2^{-N}$ being a FPN, regardless of x.

A few peculiar cases have been omitted in the sketch of this proof. Automatic proof checking allows us to trustfully guarantee that these cases have been all checked in our publicly available proof scripts. The only surprising condition is presented in this section. The other cases are easily generalized from Theorems 3 and 4. So just by wrapping these two results together, we can state the following theorem in its full length, verified with Coq.

Theorem 5 (`Fmac_arg_reduct_correct3`*):*
Assume

- $p > 3$,
- x is a p-bit FPN,
- R is a positive normal p-bit FPN,
- C_1 is the $(p - 2)$-bit FPN obtained by rounding $1/R$ to $p - 2$ bits using round-to-nearest mode,
- C_1 is not exactly a power of 2,
- $C_1 \geq 2^{p+\max(-1,N)}\lambda$,
- $z = \{3 \cdot 2^{p-N-2} + xR\}_{\mathsf{fma}} \ominus 3 \cdot 2^{p-N-2}$,
- $|xR| \leq 2^{p-N-2} - 2^{-N}$,
- 2^{-N} is a FPN.

Then $x - zC_1$ is a p-bit FPN.

In short, if C_1 is rounded to the nearest from $1/R$ with $p-2$ bits and z is computed as usual, then the fma does not make any round-off error. In Tables I and II we present constants R and C_1 for π and $\ln(2)$. These constants are for the exponential and the fast reduction phase of the trigonometric functions [1], [3], [9], [22].

The hypotheses may seem numerous and restrictive but they are not. As R and C_1 are pre-computed, the corresponding requirements can be checked beforehand. Moreover, those requirements are weak: for example with $0 <= N <= 10$ in double precision, we need $C_1 \geq 2^{-1011} \approx 4.510^{-305}$. There is no known elementary function for which C_1 ever comes near a power of 2. The only nontrivial requirement left is the bound on $|xR|$.

V. Getting More Accurate Reduced Arguments

As we pointed out in the introduction in Section I, sometimes the reduced argument $u = x - zC_1$ is not accurate enough due to the limited precision in C_1 as an approximation to C. When this happen another FPN C_2 containing the lower bits of the constant C has to be made available and the new reduced argument is now $x - zC_1 - zC_2$. Assume that the conditions of Theorem 5 hold. In particular C_1 has $p - 2$ bits in its significand.

The number $x - zC_1 - zC_2$ can be computed exactly [23] as the sum of two floats. But here because we know certain conditions on z, C_1, and C_2 as FPNs, we can do it faster. Inspired by [23], we propose the following Algorithm 5.1 to accomplish the task. It is built upon two known algorithms:

Precision	Single	Double	Double extended	Quad
R	$10680707 \cdot 2^{-25}$	$5734161139222659 \cdot 2^{-54}$	$11743562013128004906 \cdot 2^{-65}$	$66110376882906993436829972821387 30 \cdot 2^{-114}$
C_1	$13176796 \cdot 2^{-22}$	$7074237752028440 \cdot 2^{-51}$	$14488038916154245684 \cdot 2^{-62}$	$81560408330151882008337430813741 36 \cdot 2^{-111}$
C_2	$-11464520 \cdot 2^{-45}$	$4967757600021504 \cdot 2^{-105}$	$14179128828124470480 \cdot 2^{-126}$	$93516615446317514493723239679207 68 \cdot 2^{-226}$
C_3	$-15186280 \cdot 2^{-67}$	$7744522442262976 \cdot 2^{-155}$	$10700877088903390780 \cdot 2^{-189}$	$-9186378203702558149401308890796 140 \cdot 2^{-334}$

Precision	Single	Double	Double extended	Quad
R	$12102203 \cdot 2^{-23}$	$6497320848556798 \cdot 2^{-52}$	$13306513097844322492 \cdot 2^{-63}$	$74909009286315393943232627301955 14 \cdot 2^{-112}$
C_1	$11629080 \cdot 2^{-24}$	$6243314768165360 \cdot 2^{-53}$	$12786308645202655660 \cdot 2^{-64}$	$71980518562473539470808149036912 40 \cdot 2^{-113}$
C_2	$-8577792 \cdot 2^{-52}$	$-7125764960002032 \cdot 2^{-106}$	$-15596301547560248640 \cdot 2^{-130}$	$-5381235925004637553074520129202 340 \cdot 2^{-224}$
C_3	$-8803384 \cdot 2^{-72}$	$-7338834209110452 \cdot 2^{-161}$	$-13766585803531045332 \cdot 2^{-192}$	$-9437982846677142208552339635087 788 \cdot 2^{-338}$

- Fast2Mult(x,y) that computes the rounded product of x and y and its error (2 flops) [24].
- Fast2Sum(x,y) that computes the rounded sum of x and y and its error (3 flops), under the assumption that either $x = 0$, or $y = 0$, or $|x| \geq |y|$, or there exist integers n_x, e_x, n_y, e_y such that $x = n_x 2^{e_x}$ and $y = n_y 2^{e_y}$ and $e_x \geq e_y$ [10].

Algorithm 5.1 (Super accurate argument reduction):

⚠ The correctness of this algorithm is only guaranteed under the conditions of Theorem 6. It does not work with any C_1, C_2!

$$
\begin{aligned}
u &= \circ(x - zC_1), \\
v_1 &= \circ(u - zC_2), \\
(p_1, p_2) &= \text{Fast2Mult}(z, C_2), \\
(t_1, t_2) &= \text{Fast2Sum}(u, -p_1), \\
v_2 &= \circ(\circ(\circ(t_1 - v_1) + t_2) - p_2).
\end{aligned}
$$

Theorem 6 (`FArgReduct4.v` *file):* Assume
- $p > 4$,
- x is a p-bit FPN,
- R is a positive normal p-bit FPN,
- C_1 is the $(p - 2)$-bit FPN obtained by rounding $1/R$ to $p - 2$ bits using round-to-nearest mode,
- C_1 it is not exactly a power of 2,
- $z = \left\{3 \cdot 2^{p-N-2} + xR\right\}_{\text{fma}} \ominus 3 \cdot 2^{p-N-2}$,
- $|xR| \leq 2^{p-N-2} - 2^{-N}$,
- 2^{-N} is a normal p-bit FPN,
- $C_1 \geq 2^{p+\max(-1, p+N-2)}\lambda$,
- C_2 is a FPN and an integer multiple of $8\text{ulp}^{\circ 2}(C_1)$,
- $|C_2| \leq 4\text{ulp}(C_1)$,
- v_1 and v_2 are computed using Algorithm 5.1.

Then Fast2Sum works correctly and we have the mathematical equality $v_1 + v_2 = x - zC_1 - zC_2$ (all the computations of the last line indeed commit no rounding errors).

The first requirements are very similar to the previous ones. The "no underflow" bound on C_1 has been raised, but is still easily achieved by real constants. For a typical N between 0 and 10 used by the existing elementary math libraries in IEEE double precision, it suffices that $C \geq 10^{-288}$.

The most important add-ons are the requirements on C_2: it must be much smaller than C_1 (it is near the difference between the constant C and C_1). And C_2 must not be "too precise". In fact, $C_1 + C_2$ will have $2p - 4$ bits as shown in Figure 7. If by chance, there are a lot of zeroes just after C_1, we cannot take advantage of that to get a more precise C_2. This is a real drawback, but it does not happen very often that many zeroes are just at the inconvenient place.
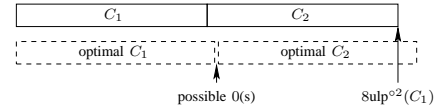

Fig. 7. Respective layouts of our C_1 and C_2 compared to optimal values.

This algorithm may seem simple but it is a very powerful tool. It is *exact* and it is very *fast*: the generic algorithm [23] costs 20 flops while this one costs only 9 flops! More, the result is more usable than expected as it fits in only one float instead of two in the general case.

As for the computation of C_2, the requirements are rather low: there are several C_2 fulfilling them. It may be useful to choose one of them in order to have the bigger or the smaller C_2 possible. Algorithm 5.2 gives one way to compute a convenient C_2.

The idea of the proof for Theorem 6 is a careful study of the possible exponents for the involved FPNs. We first prove that x is an integer multiple of $2^{-N}\text{ulp}(C_1)$. This is done for whether z is 2^{-N} or not to guarantee the correctness of Fast2Sum.

We then prove that $t_1 - v_1$ fits in a FPN. This proof is obtained by noticing that t_1 and v_1 are integer multiples of

$2^{-N-1}\text{ulp}^{\circ 2}(C_1)$ and that that $|t_1 - v_1| < 2^{p-N-1}\text{ulp}^{\circ 2}(C_1)$.

The next step is about $t_1 - v_1 + t_2 = u - p_1 - v_1$ being a FPN. We do it similarly as all these quantities are also integer multiples of $2^{-N-1}\text{ulp}^{\circ 2}(C_1)$ and as we easily have that $|t_1 - v_1 + t_2| < 2^{p-N-1}\text{ulp}^{\circ 2}(C_1)$.

We finally prove that $t_1 - v_1 + t_2 - p_2 = u - zC_2 - v_1$ fits in a FPN. Its least significant non-zero bit is at most shifted N times down compared to the least significant non-zero bit of C_2. For this reason, we require that C_2 is an integer multiple of $8\text{ulp}^{\circ 2}(C_1)$.

This proof needs a careful study of the relationships between the various floats and their exponent values. The formal proof and its genericity allowed us a better understanding of the respective layouts of the FPNs, that is the key of the correctness of Algorithm 5.1.

Algorithm 5.2 (Computation of C_2): Let C be the exact constant (for example, π or $\ln 2$).

$$\begin{aligned} R &= \circ_p(1/C), \\ C_1 &= \circ_{p-2}(1/R), \end{aligned}$$

and take C_2 to be the first many significand bits of $C - C_1$ so that its least non-zero bit must be greater than or equal to $\log_2(\text{ulp}(C_1)) - p + 4 = \log_2\big(8\text{ulp}^{\circ 2}(C_1)\big)$, e.g.,

$$C_2 = \left\lceil \frac{(C - C_1)}{8\text{ulp}^{\circ 2}(C_1)} \right\rfloor 8\text{ulp}^{\circ 2}(C_1),$$

where $\lceil \cdot \rfloor$ is one of the round-to-integer operations.

This C_2 has all the expected properties except that we do not know for sure if $|C_2| \le 4\text{ulp}(C_1)$. Note that C_1 is not gotten by directly rounding C but rather $C_1 = \circ\big(1/\circ\big(\frac{1}{C}\big)\big)$.

Theorem 7 (`gamma2_le`): Assume
- $p > 3$,
- C is a real positive constant,
- R is the p-bit FPN obtained by rounding $1/C$ to p bits using round-to-nearest mode,
- R is a positive normal p-bit FPN,
- C_1 is the $(p-2)$-bit FPN obtained by rounding $1/R$ to $p-2$ bits using round-to-nearest mode,
- C_1 is not exactly a power of 2,
- $C_1 \ge 2^{p-1}\lambda$.

Then $|C - C_1| \le 4\text{ulp}(C_1)$.

As C is not too far from C_1, we have that $C \le 2^{p+1}\text{ulp}(C_1)$. We now bound $C - C_1$:
$|C - C_1| \le |C - 1/R| + |1/R - C_1| \le \frac{C}{R}|R - 1/C| + |1/R - C_1|$, so $|C - C_1| \le \frac{C}{R}\text{ulp}(R)/2 + 4\text{ulp}(C_1)/2 \le C2^{-p-1} + 2\text{ulp}(C_1)$, hence the result.

This means that the formula for C_2 given above yields a FPN fulfilling the requirements of Theorem 6.

VI. CONCLUSIONS

We have presented Coq verified theorems that prove the correctness and effectiveness of a much faster technique based on the commonly used argument reduction in elementary function computations, on machines that have hardware support for fused-multiply-add instructions. The conditions of these theorems are easily met as our analysis indicates. While we have showed it is not always possible to use the most accurate parameters under all circumstances, an almost best possible selection can be used at all times: to zero out the last 2 bits.

We have presented also a very accurate second step argument reduction. We provide a way to compute C_2 which is not the most precise possible, but is usually 2 bits away from it (and can be rounded as needed by the programmer). The most interesting part is the possibility to compute with FPNs the exact error of the second step of the argument reduction and the fact that this error is exactly representable by only one FPN. It makes the third step unexpectedly easy as we have a mathematical equality between the computed FPNs and a very good approximation of $x - zC$ (with a known error).

Except for the computation of C_2, all the rounding used should be rounding to nearest, ties to even. But our proofs are generic enough to show that our results still hold when using rounding to nearest, where cases of ties can be decided in any coherent way [25]. This includes rounding to nearest, ties away from zero that is found in the revision of the IEEE-754 standard.

The formal verification forces us to provide many tedious details in the proofs but gives us a guarantee on our results. The proposed theorems are sufficient in the sense that effective parameters for efficient argument reductions can be obtained without any difficulty.

Our theorems provides us with sufficient conditions for $x - zC_1$ to be a FPN. This means that $x - zC_1$ could be a FPN even when one or more of the conditions fails for some specific values of C, C_1 and R as published in the past [1], [7]. We may work on this in the future even though there is only a limited space for improvement as only the last two bits of C_1 can be changed to make the constant more accurate.

The algorithms proved can be applied to any floating-point format (IEEE single, double or extended for example). Intuitively, the correctness of these algorithms should come as natural. Nevertheless, rigorous proofs are not trivial due to a few special cases that could have been easily dismissed by hand-waving proofs.

REFERENCES

[1] P. Markstein, *IA-64 and elementary functions: speed and precision.* Prentice Hall, 2000.

[2] N. Brisebarre, D. Defour, P. Kornerup, J.-M. Muller, and N. Revol, "A new range-reduction algorithm," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 331–339, 2005.

[3] J.-M. Muller, *Elementary functions, algorithms and implementation.* Birkhauser, 2006. [Online]. Available: http://www.springer.com/west/home/birkhauser/computer+science?SGWID=4-40353-22

[4] R.-C. Li, "Near optimality of Chebyshev interpolation for elementary function computations," *IEEE Trans. Comput.*, vol. 53, no. 6, pp. 678–687, 2004.

[5] W. J. Cody and W. Waite, *Software manual for elementary functions.* Prentice Hall, 1980.

[6] P. W. Markstein, "Computation of elementary functions on the IBM RISC System/6000 processor," *IBM Journal of Research and Development*, vol. 34, no. 1, pp. 111–119, 1990. [Online]. Available: http://www.research.ibm.com/journal/rd/341/ibmrd3401N.pdf

[7] S. Story and P. T. P. Tang, "New algorithms for imporved transcendental function on IA-64," in *Proceedings of the 14th Symposium on Computer Arithmetic*, I. Koren and P. Kornerup, Eds., Adelaide, Australia, 1999, pp. 4–11. [Online]. Available: http://computer.org/proceedings/arith/0116/0116toc.htm

[8] R.-C. Li, S. Boldo, and M. Daumas, "Theorems on efficient argument reductions," in *Proceedings of the 16th Symposium on Computer Arithmetic*, J.-C. Bajard and M. Schulte, Eds., Santiago de Compostela, Spain, 2003, pp. 129–136. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00156244

[9] W. Kahan, "Minimizing $q \times m - n$," 1983, published on the net. [Online]. Available: http://www.cs.berkeley.edu/~wkahan/testpi/nearpi.c

[10] M. Daumas, L. Rideau, and L. Théry, "A generic library of floating-point numbers and its application to exact computing," in *14th International Conference on Theorem Proving in Higher Order Logics*, Edinburgh, Scotland, 2001, pp. 169–184. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00157285

[11] Y. Bertot and P. Castéran, *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*, ser. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[12] V. A. Carreño and P. S. Miner, "Specification of the IEEE-854 floating-point standard in HOL and PVS," in *1995 International Workshop on Higher Order Logic Theorem Proving and its Applications*, Aspen Grove, Utah, 1995, supplemental proceedings. [Online]. Available: http://shemesh.larc.nasa.gov/fm/ftp/larc/vac/hug95.ps

[13] D. M. Russinoff, "A mechanically checked proof of IEEE compliance of the floating point multiplication, division and square root algorithms of the AMD-K7 processor," *LMS Journal of Computation and Mathematics*, vol. 1, pp. 148–200, 1998. [Online]. Available: http://www.onr.com/user/russ/david/k7-div-sqrt.ps

[14] J. Harrison, "Floating point verification in HOL light: the exponential function," University of Cambridge Computer Laboratory, Technical Report 428, 1997. [Online]. Available: http://www.cl.cam.ac.uk/users/jrh/papers/tang.ps.gz

[15] ——, "Formal verification of floating point trigonometric functions," in *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, W. A. Hunt and S. D. Johnson, Eds., Austin, Texas, 2000, pp. 217–233.

[16] P. H. Sterbenz, *Floating point computation*. Prentice Hall, 1974.

[17] D. Goldberg, "What every computer scientist should know about floating point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–47, 1991. [Online]. Available: http://doi.acm.org/10.1145/103162.103163

[18] S. Boldo and M. Daumas, "Properties of two's complement floating point notations," *International Journal on Software Tools for Technology Transfer*, vol. 5, no. 2-3, pp. 237–246, 2004. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00157268

[19] D. Stevenson *et al.*, "An American national standard: IEEE standard for binary floating point arithmetic," *ACM SIGPLAN Notices*, vol. 22, no. 2, pp. 9–25, 1987.

[20] W. J. Cody, R. Karpinski, *et al.*, "A proposed radix and word-length independent standard for floating point arithmetic," *IEEE Micro*, vol. 4, no. 4, pp. 86–100, 1984.

[21] *Pentium Pro Family: Developer's Manual*, Intel, 1996, programmer's Reference Manual. [Online]. Available: ftp://download.intel.com/design/pro/manuals/24269101.pdf

[22] K. C. Ng, "Argument reduction for huge arguments: good to the last bit," 1992, work in progress. [Online]. Available: http://www.validgh.com/arg.ps

[23] S. Boldo and J.-M. Muller, "Some functions computable with a fused-mac," in *Proceedings of the 17th Symposium on Computer Arithmetic*, P. Montuschi and E. Schwarz, Eds., Cape Cod, Massachusetts, 2005, pp. 52–58. [Online]. Available: http://csdl.computer.org/comp/proceedings/arith/

[24] A. H. Karp and P. Markstein, "High-precision division and square root," *ACM Transactions on Mathematical Software*, vol. 23, no. 4, pp. 561–589, Dec. 1997.

[25] J. F. Reiser and D. E. Knuth, "Evading the drift in floating point addition," *Information Processing Letters*, vol. 3, no. 3, pp. 84–87, 1975.

APPENDIX

Theorem 4 can be used for any value of $2 \leq q \leq p - 1$. In most case, users are interested for the smallest possible value of q because that will give a more accurate C_1 and consequently a more accurate reduced argument. For this reason, we proved Theorem 5 for $q = 2$. The following theorem is under the hypothesis that

$$RC_1 \leq 1,$$

while $2 \leq q \leq p - 1$ still. This add-on is enough to guarantee cases that are left over by Theorem 4.

Theorem 8 (`Fmac_arg_reduct_correct2`*):*
Assume

- $p > 3$,
- $2 \leq q < p - 1$,
- x is p-bit FPN,
- R is a positive normal p-bit FPN,
- C_1 is the $(p - q)$-bit FPN obtained by rounding $1/R$ to $p - q$ bits using round-to-nearest mode,
- C_1 is not exactly a power of 2,
- $C_1 \geq 2^{p-q+\max(1,N-1)}\lambda$,
- $z = \left\{ 3 \cdot 2^{p-N-2} + xR \right\}_{\mathsf{fma}} \ominus 3 \cdot 2^{p-N-2}$,
- 2^{-N} is a FPN,
- $|xR| \leq 2^{p-N-2} - 2^{-N}$,
- $RC_1 \leq 1$.

Then $x - zC_1$ is a p-bit FPN.

We essentially need to consider how to make R and C_1 satisfy this new constraint. Since there is no strict connection between R, C_1 on one hand and C on the other hand, we can either use R to be the correctly rounded FPN nearest $1/C$ or we may alternatively add or subtract one or a few ulps so that the additional inequality is met.